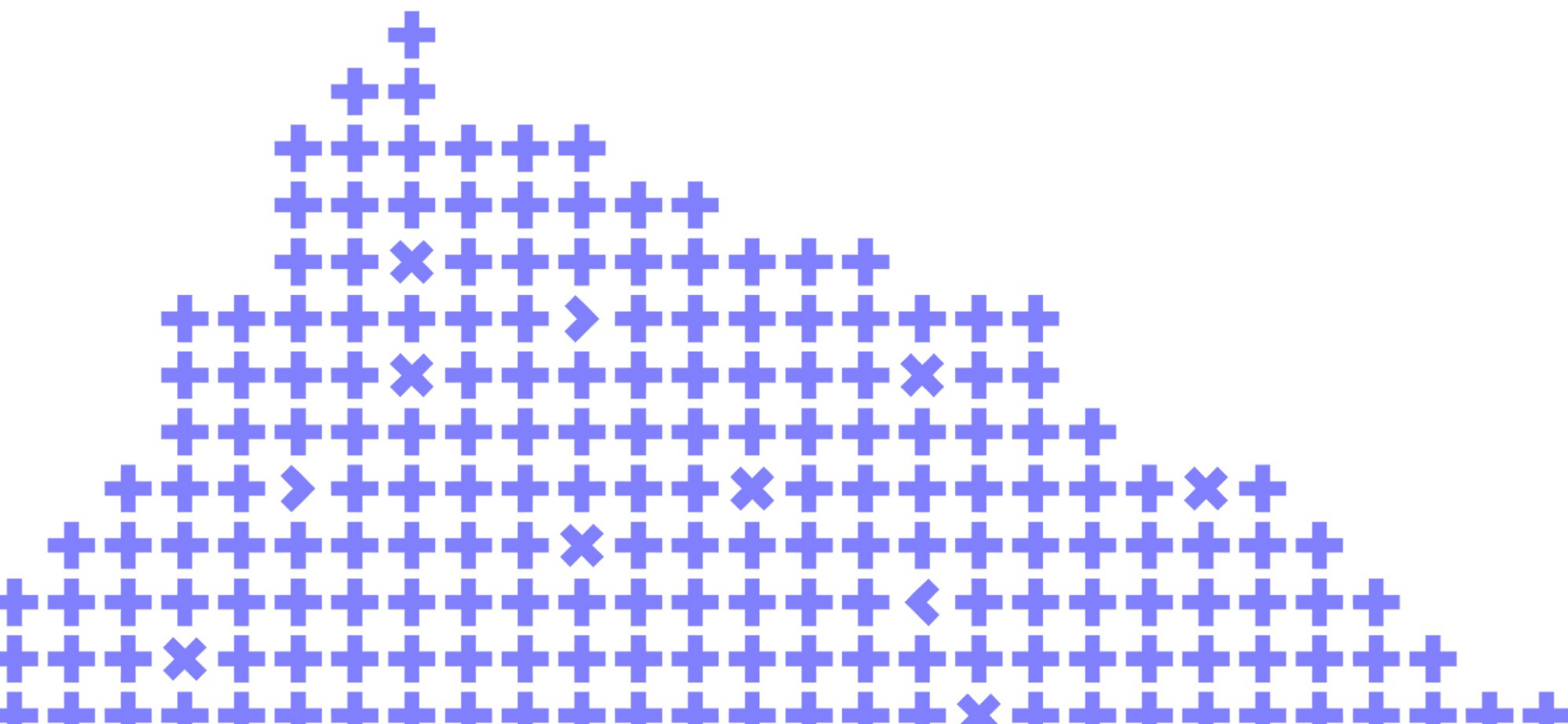


Yandex Query – Serverless Federated Query System. Inside View

Aleksandr Khoroshilov



Co-organizer

Yandex

Agenda

1. What is Yandex Query?

2. Big data processing

3. Multitenant architecture

4. Capacity and isolation

5. CPU usage limitation

6. Backup and recovery

What is Yandex Query?

1/2

Yandex Query is

- Serverless (available in Yandex Cloud)
- Federated (mix of different data providers)
- Query System (SQL-like)

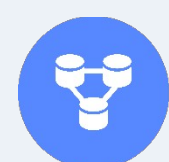
Stream and batch processing
with uniform syntax for easy
development and debugging

For Data Engineers and Analysts

Uniform syntax increases productivity



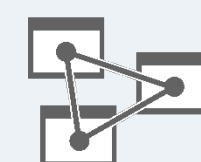
Yandex Query



YDB



Amazon
Athena



Azure Analysis Services



Apache
Flink

What is Yandex Query?

2/2

Big data

- Data is stored in the cloud
- Parallel processing
- Integrated with other services

```
1 SELECT message, CAST(microseconds as UINT64) as microseconds
2 FROM bindings.`logs`
3 WHERE microseconds > CurrentUtcTimestamp() - Interval("PT24H")
4 AND priority = "ERROR"
5 ORDER BY microseconds DESC;
```

Run Validate Explain Save Clone

Completed Run 10 December 2022, 23:25:50 00:20

Result Plan AST Monitoring Statistics Meta

Result (55)

#	message ⓘ	microseconds ⓘ
1	"Create rate limiter resource \"b1g918jf99v96n47o7u6\" error: BAD_REQUEST { <main>: Fatal: Resource already exists and has different settings. }"	1670628773594231
2	"Create rate limiter resource \"b1g918jf99v96n47o7u6\" error: BAD_REQUEST { <main>: Fatal: Resource already exists and has different settings. }"	1670628771107286
3	"Create rate limiter resource \"b1g918jf99v96n47o7u6\" error: BAD_REQUEST { <main>: Fatal: Resource already exists and has different settings. }"	1670628725682844

Dogfooding to improve the service quality

Big data processing







1/4

Logs are stored in Object Storage

- Large number of files (objects)
- Grouped by a key prefix (virtual folders)
- Records vary in size and per prefix distribution

Object Storage / ... / yq-streams-logs / hot / year=2022 / month=12 / day=08 / hour=21

Object name

<input type="checkbox"/>	Name	Size	Storage class	Last change	
<input type="checkbox"/>	 yq-logs_0-6260764_6260816.raw.gz	14.98 KB	Standard	08.12.2022, at 21:01	...
<input type="checkbox"/>	 yq-logs_0-6260817_6260894.raw.gz	28.17 KB	Standard	08.12.2022, at 21:03	...
<input type="checkbox"/>	 yq-logs_0-6260895_6260976.raw.gz	24.47 KB	Standard	08.12.2022, at 21:05	...
<input type="checkbox"/>	 yq-logs_0-6260977_6261048.raw.gz	11.94 KB	Standard	08.12.2022, at 21:07	...
<input type="checkbox"/>	 yq-logs_0-6261049_6261121.raw.gz	17.9 KB	Standard	08.12.2022, at 21:09	...
<input type="checkbox"/>	 yq-logs_0-6261122_6261197.raw.gz	15.93 KB	Standard	08.12.2022, at 21:11	...

Fast processing requires parallel execution on multiple nodes

Big data processing







2/4

Naïve solution

- List objects and apply filters (prune)
- Send subsets of keys to workers (map)
- Download and process data (reduce)

Object Storage / ... / yq-streams-logs / hot / year=2022 / month=12 / day=08 / hour=21

Object name

<input type="checkbox"/>	Name	Size	Storage class	Last change	
<input type="checkbox"/>	 yq-logs_0-6260764_6260816.raw.gz	14.98 KB	Standard	08.12.2022, at 21:01	...
<input type="checkbox"/>	 yq-logs_0-6260817_6260894.raw.gz	28.17 KB	Standard	08.12.2022, at 21:03	...
<input type="checkbox"/>	 yq-logs_0-6260895_6260976.raw.gz	24.47 KB	Standard	08.12.2022, at 21:05	...
<input type="checkbox"/>	 yq-logs_0-6260977_6261048.raw.gz	11.94 KB	Standard	08.12.2022, at 21:07	...
<input type="checkbox"/>	 yq-logs_0-6261049_6261121.raw.gz	17.9 KB	Standard	08.12.2022, at 21:09	...
<input type="checkbox"/>	 yq-logs_0-6261122_6261197.raw.gz	15.93 KB	Standard	08.12.2022, at 21:11	...

Listing is the bottleneck – can we do it better?

Big data processing

3/4

Better solution

- List key prefixes and apply filters (prune)
- Send subsets of prefixes to workers (map)
- List keys by prefixes (expand)
- Download and process data (reduce)

Object Storage / ... / yq-streams-logs / hot / year=2022 / month=12 / day=08

Object name

<input type="checkbox"/>	Name	Size	Storage class	Last change	⚙
<input type="checkbox"/>	📁 hour=00	—	—	—	...
<input type="checkbox"/>	📁 hour=01	—	—	—	...
<input type="checkbox"/>	📁 hour=02	—	—	—	...
<input type="checkbox"/>	📁 hour=03	—	—	—	...
<input type="checkbox"/>	📁 hour=04	—	—	—	...
<input type="checkbox"/>	📁 hour=05	—	—	—	...

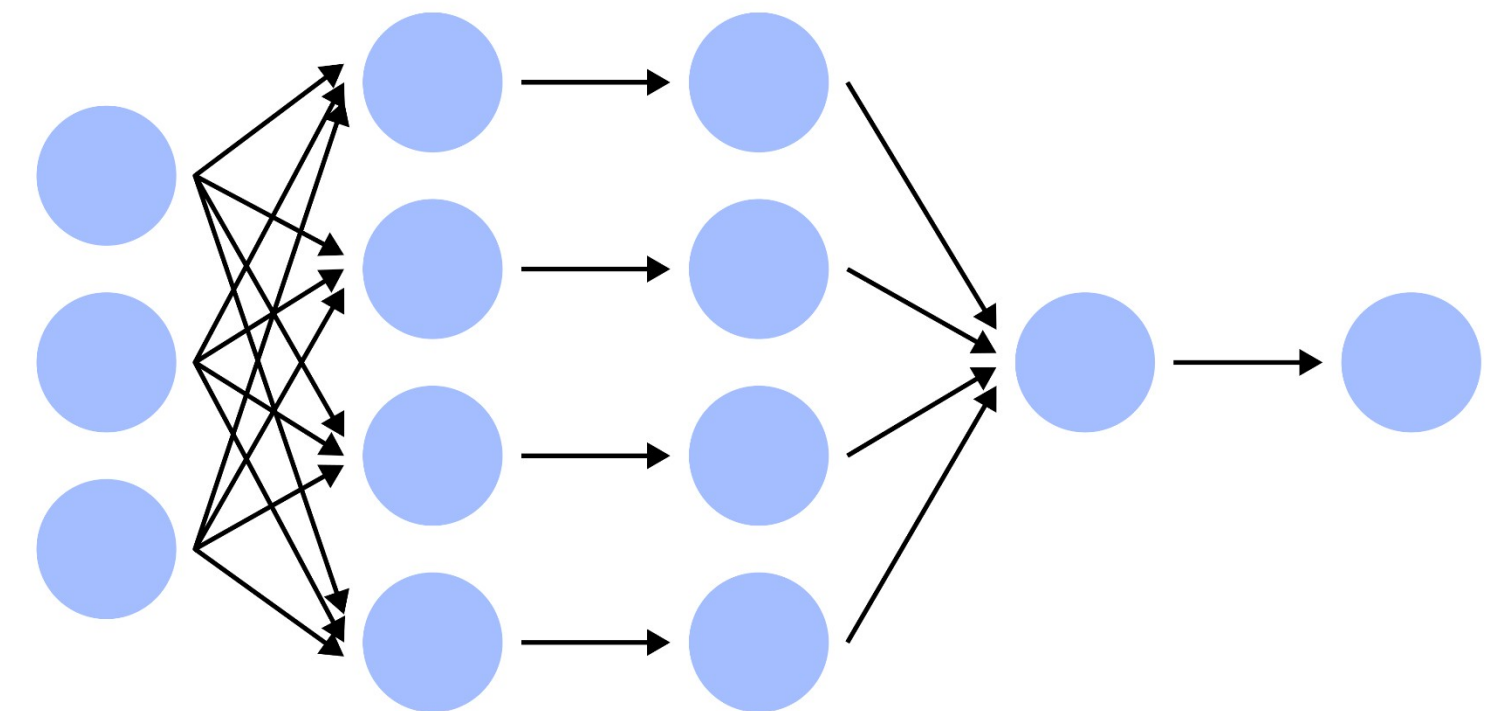
Listing is also parallelized, but can we do it even better?

Big data processing

4/4

Extreme optimization (even load)

- Shuffle keys with back pressure
- Split single-item processing (if format supports)



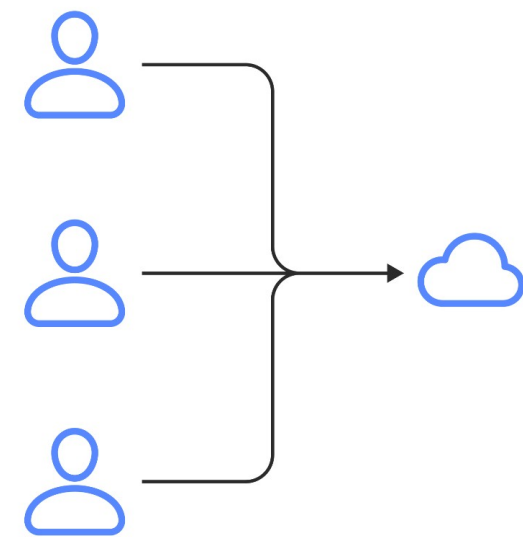
In an ideal system all nodes start and finish simultaneously

Multitenant Architecture

1/2

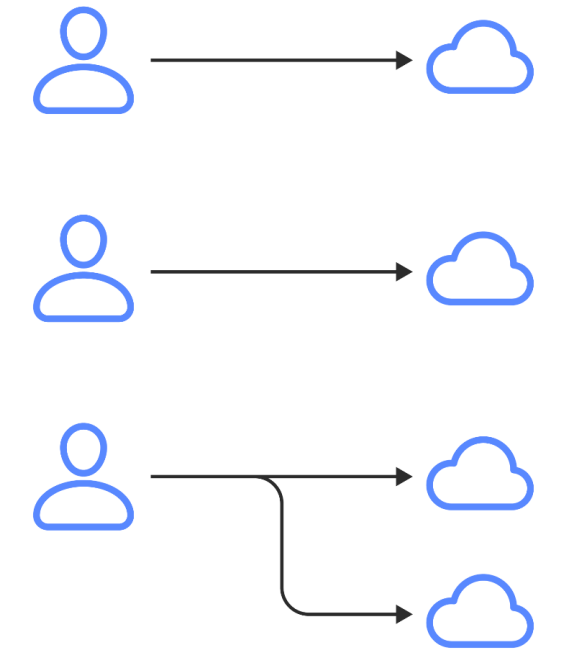
Multuser/Multitenant

- Shared environment
- Elastic
- Pay as you go



Managed/On Premise

- Isolated environment
- Fixed size cluster
- Constant costs of ownership



Multitenant architecture is a common choice for cloud services

Multitenant Architecture

2/2

Advantages

- CPU utilization
- Low support costs
- Scalability

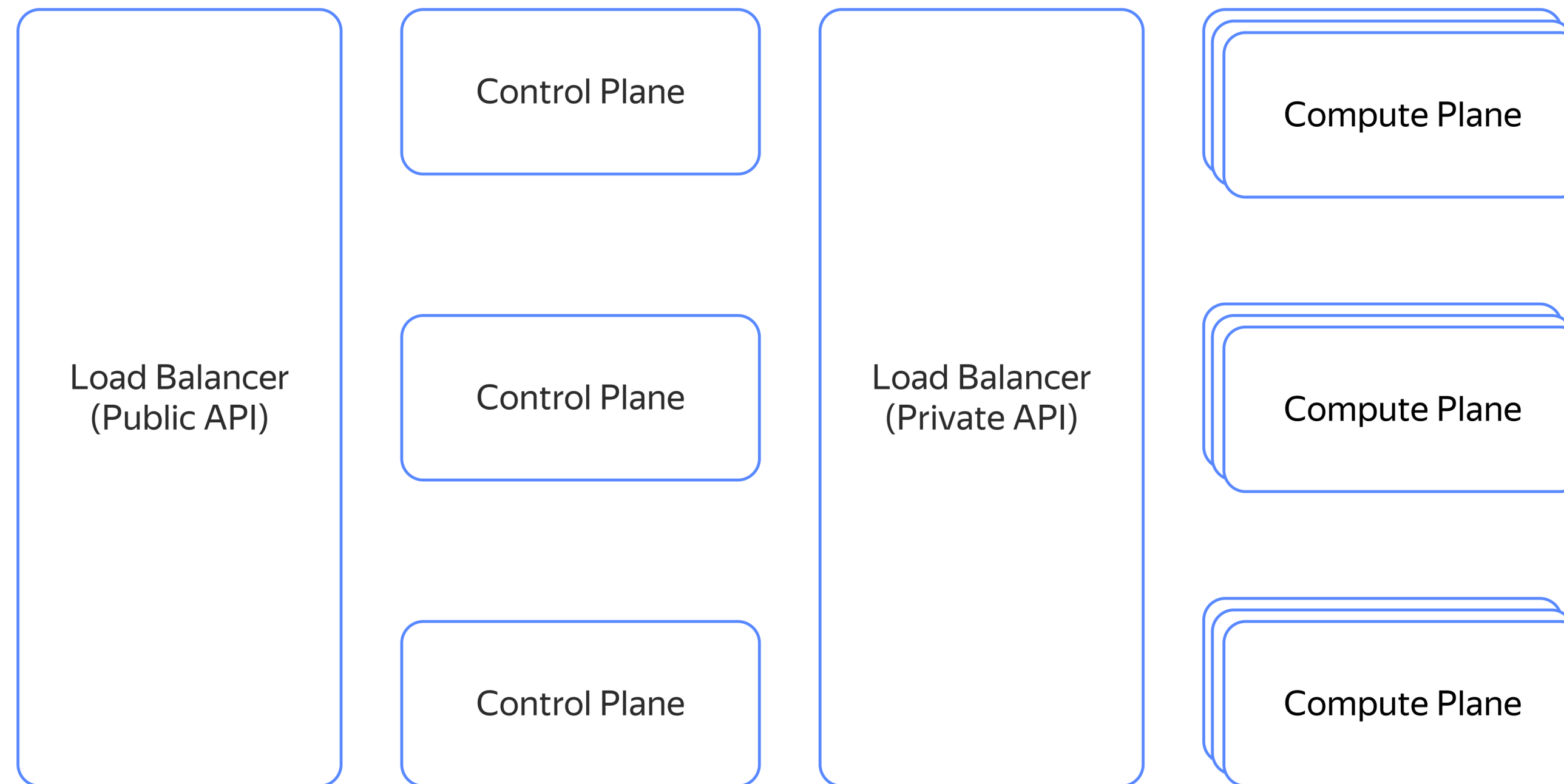
Disadvantages

- Lower level of isolation
- Higher failure probability
- Security issues

Multitenant architecture is flexible and complex in design

Capacity and isolation

1/3

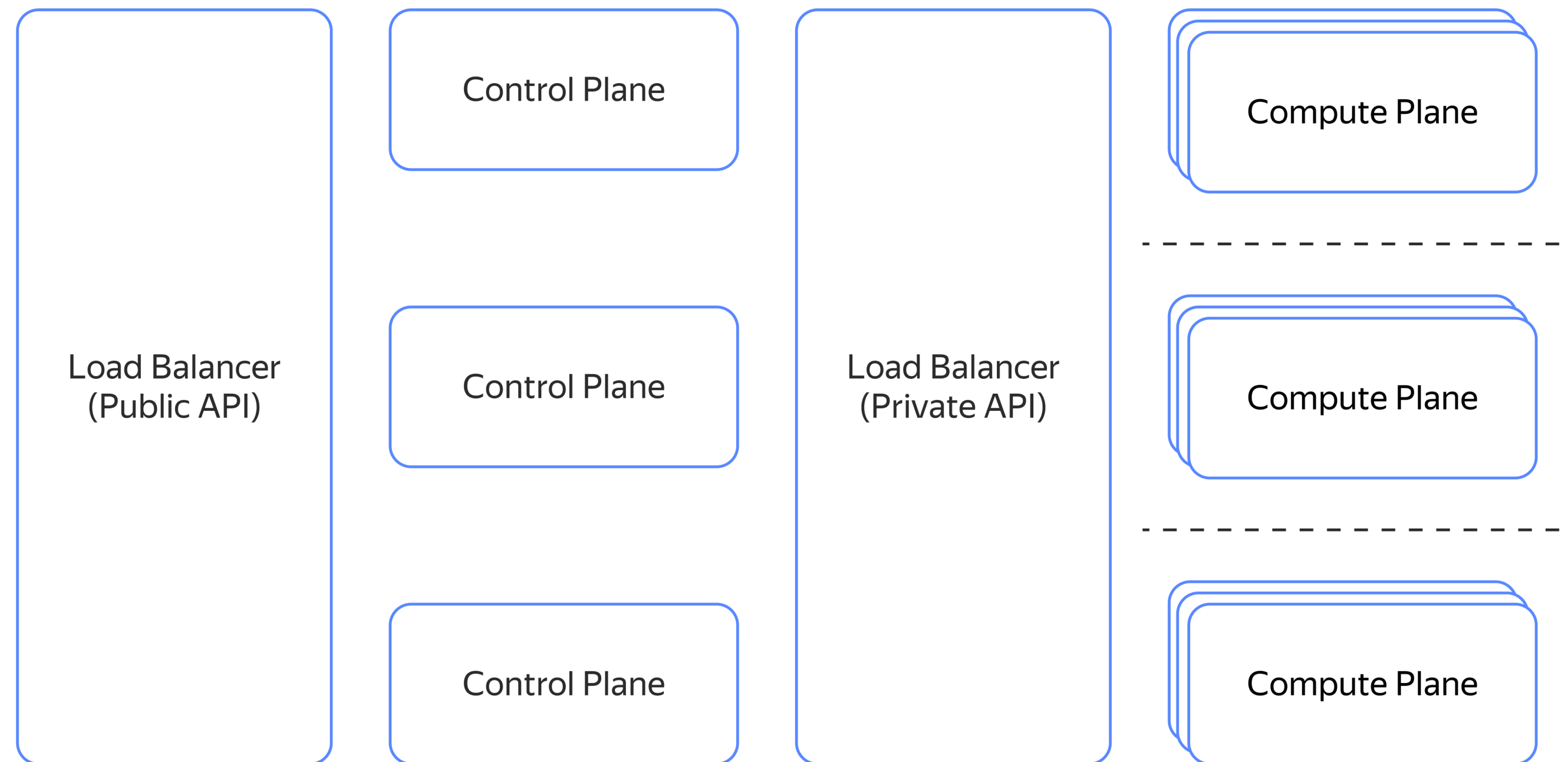


Compute plane is isolated from the control plane

Larger computing power, lower degradation in case of a node failure

Capacity and isolation

2/3

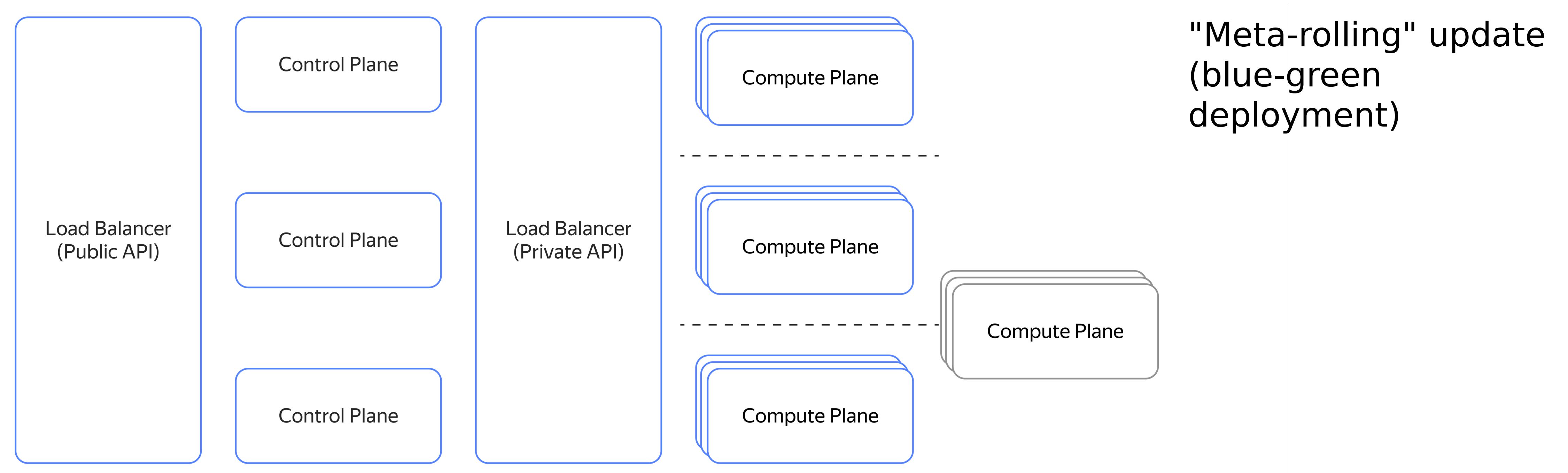


Multiple tenants
to reduce blast radius

Split the compute plane into isolated parts

Capacity and isolation

3/3



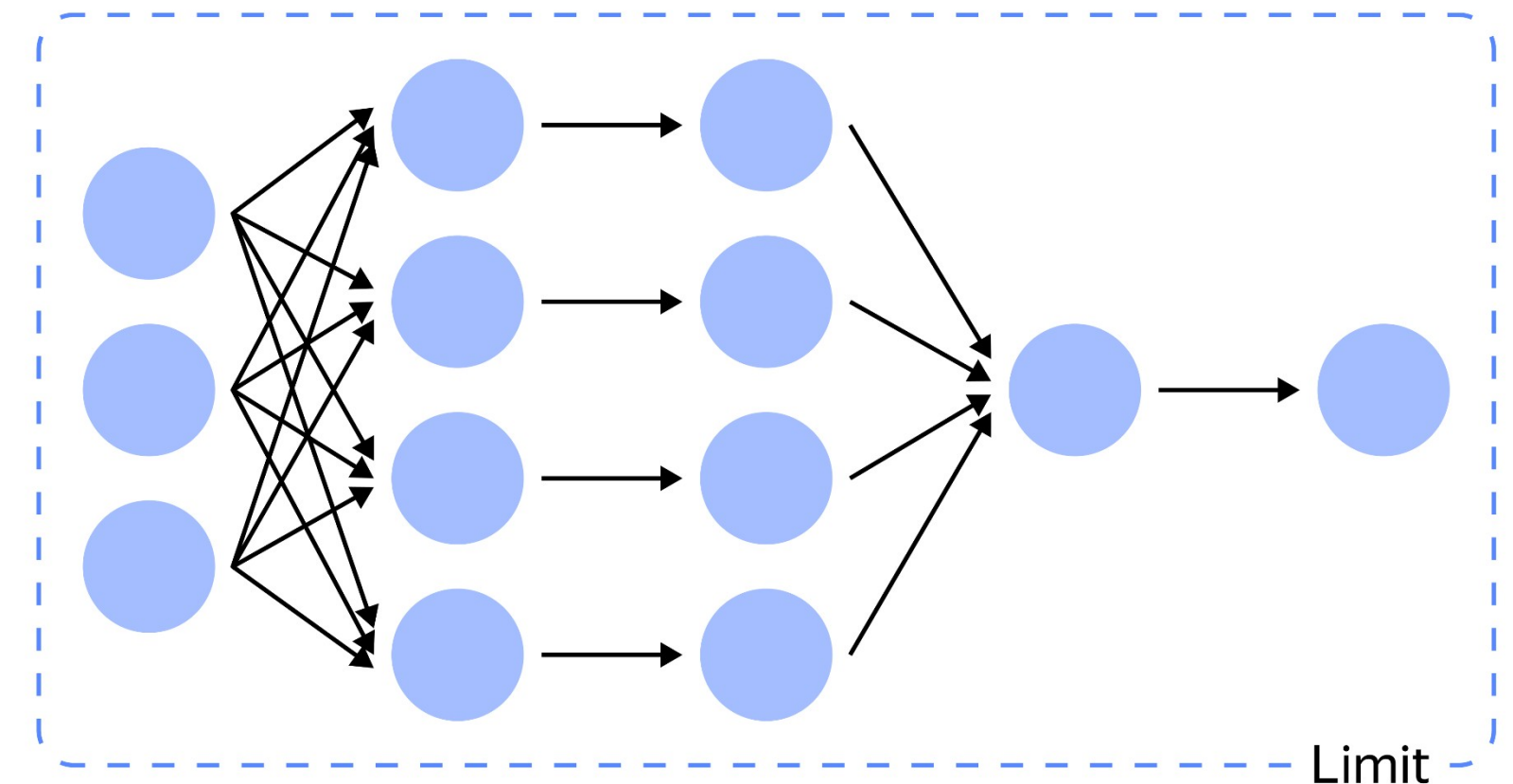
Instant redirect for zero downtime with a small overhead (less than x2)

CPU usage limitation

1/3

Query is processed on multiple nodes

- Total CPU (per request) is limited
- Average load over the time is measured
- Performance is not affected



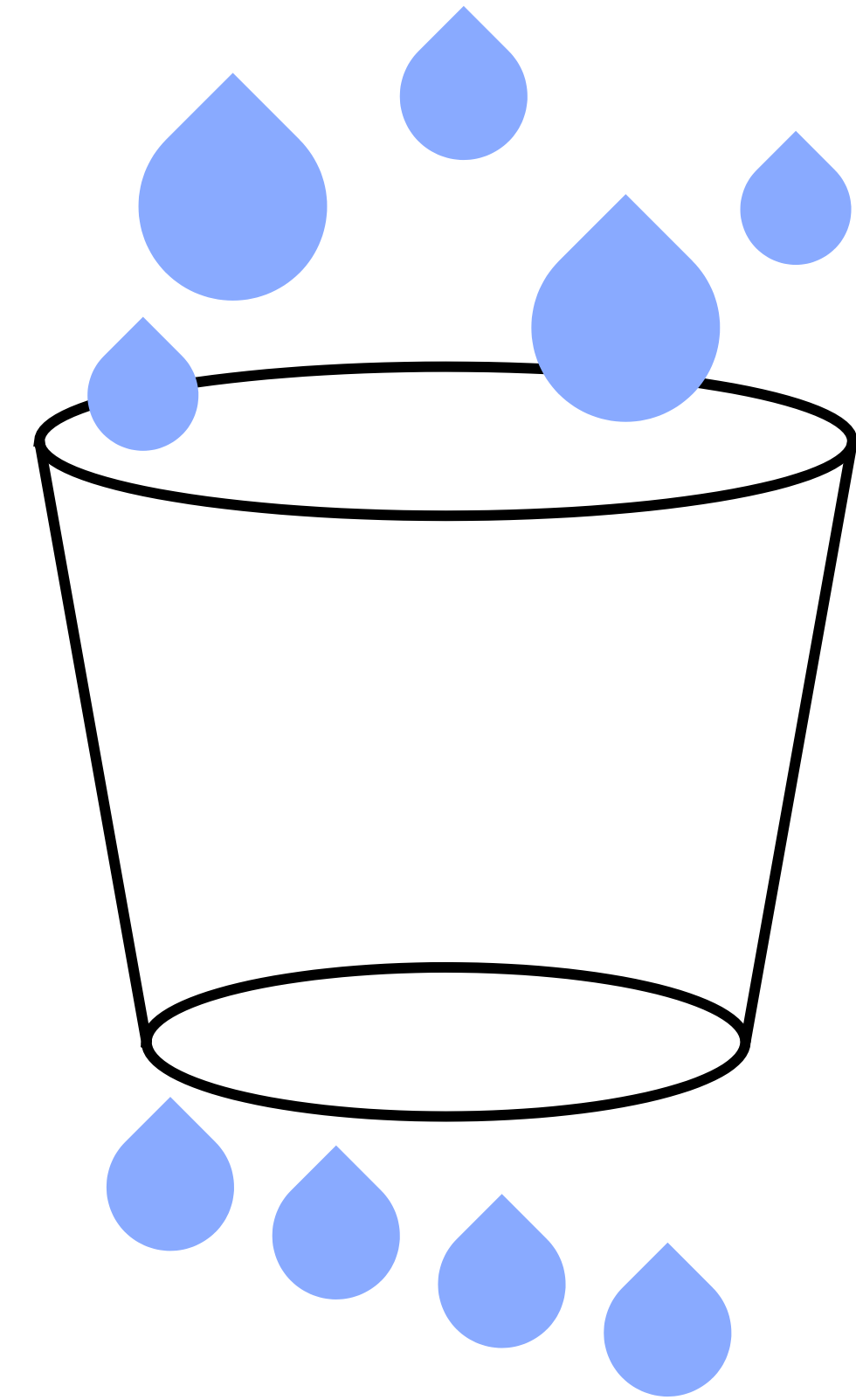
Usage of the shared resources in a multiuser system must be limited

CPU usage limitation

2/3

Leaky bucket

- Each request "fills" the bucket
- Requests vary in size and are not aligned in time
- Bucket "leaks" at a fixed rate
- If bucket is full, incoming requests are delayed



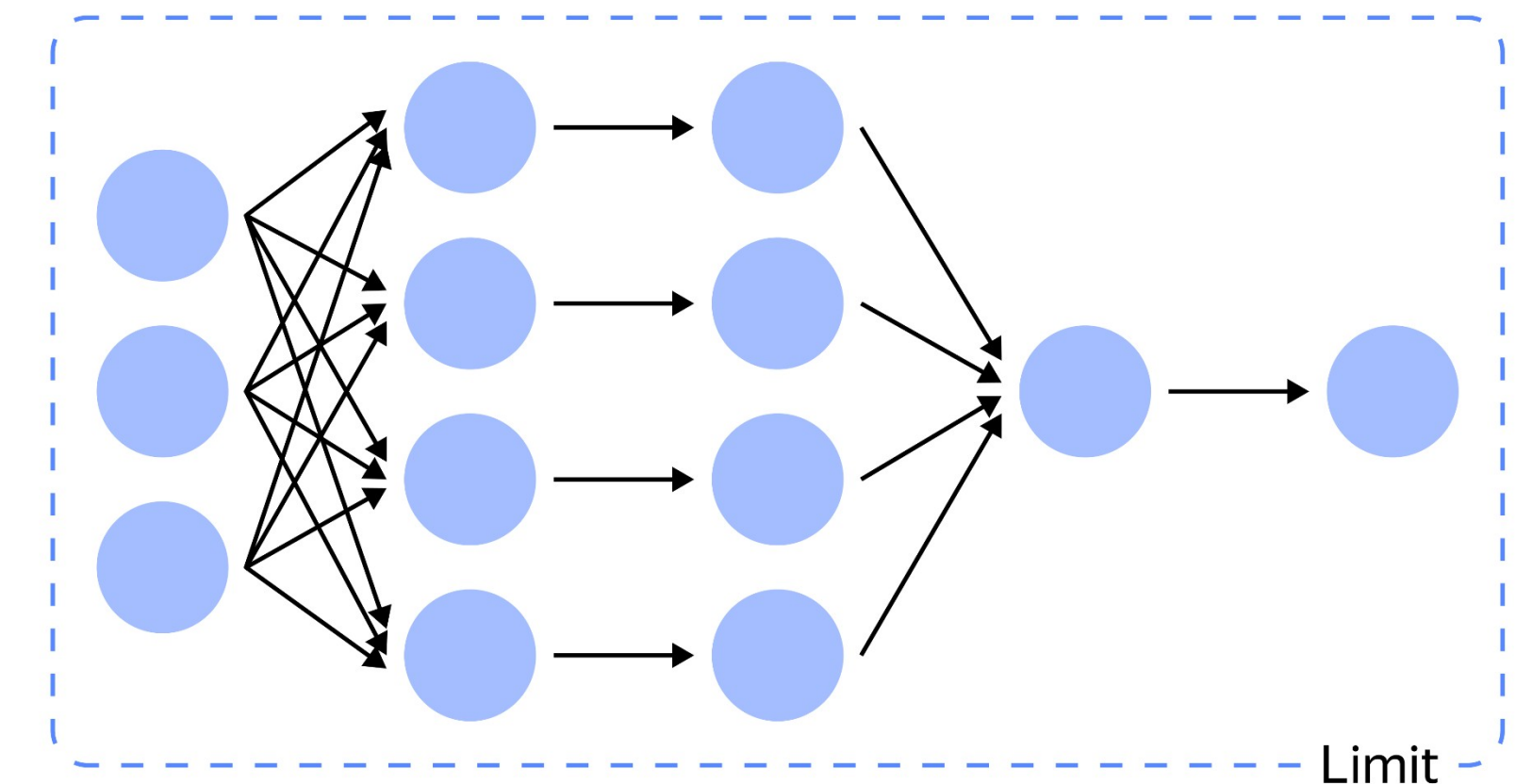
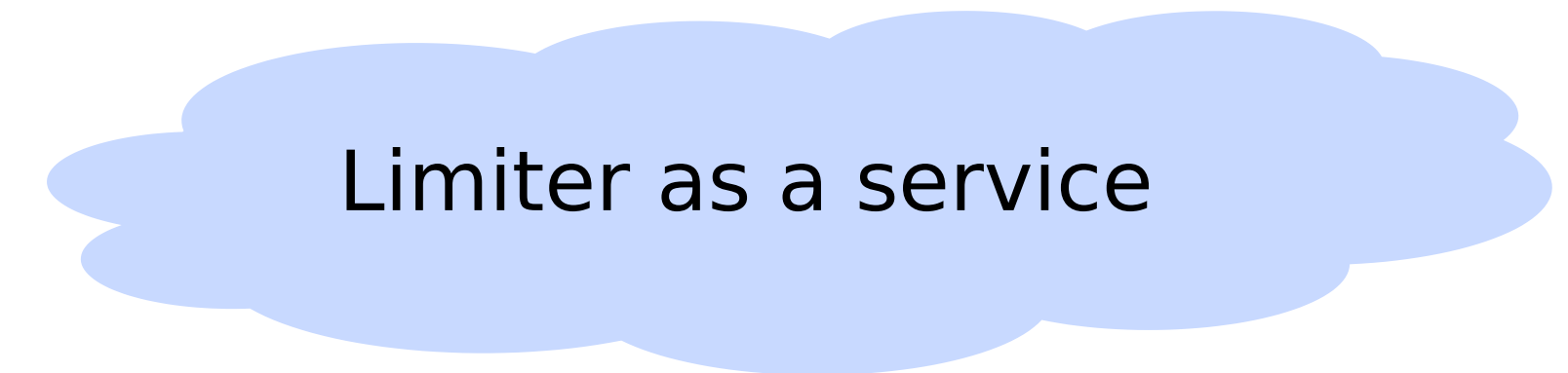
Classic rate-limiting algorithm

CPU usage limitation

3/3

Distributed version

- Limiter is a fault tolerant external service
- Each worker requests the quota in advance
- Local overbudget CPU usage is allowed



Limiting should not affect performance

Performance and reliability

1/4

Resource (CPU, memory, dataflow) usage control

- Adjustable quotas
- Hard limits

To mitigate

- Misbehaved queries
- Large bills
- Intentional DDOS attacks

A lot of computing power is not always desirable

Performance and reliability

2/4

Higher number of node failures as a result of

- Large number of nodes
- Mix of queries from different users

System must provide expected behavior

- Failovers and retries
At least once
- What about exactly once?
In great demand, but causes extra overhead

Large number of nodes increases failure (of a single node) probability

Performance and reliability

3/4

User can get an exactly-once processing "under certain conditions"

- At least once with deduplication
Like UPSERT, requires determinacy
- Tagged data with non-transactional data providers
With background GC
- 2PC if a data provider supports transactions

Exactly-once processing may affect performance

Performance and reliability

4/4

Stream processing is checkpointed

- System periodically saves the state of the request (checkpoint)
- Failures are restarted from the last checkpoint
- Checkpoints are stored in the transactional DB

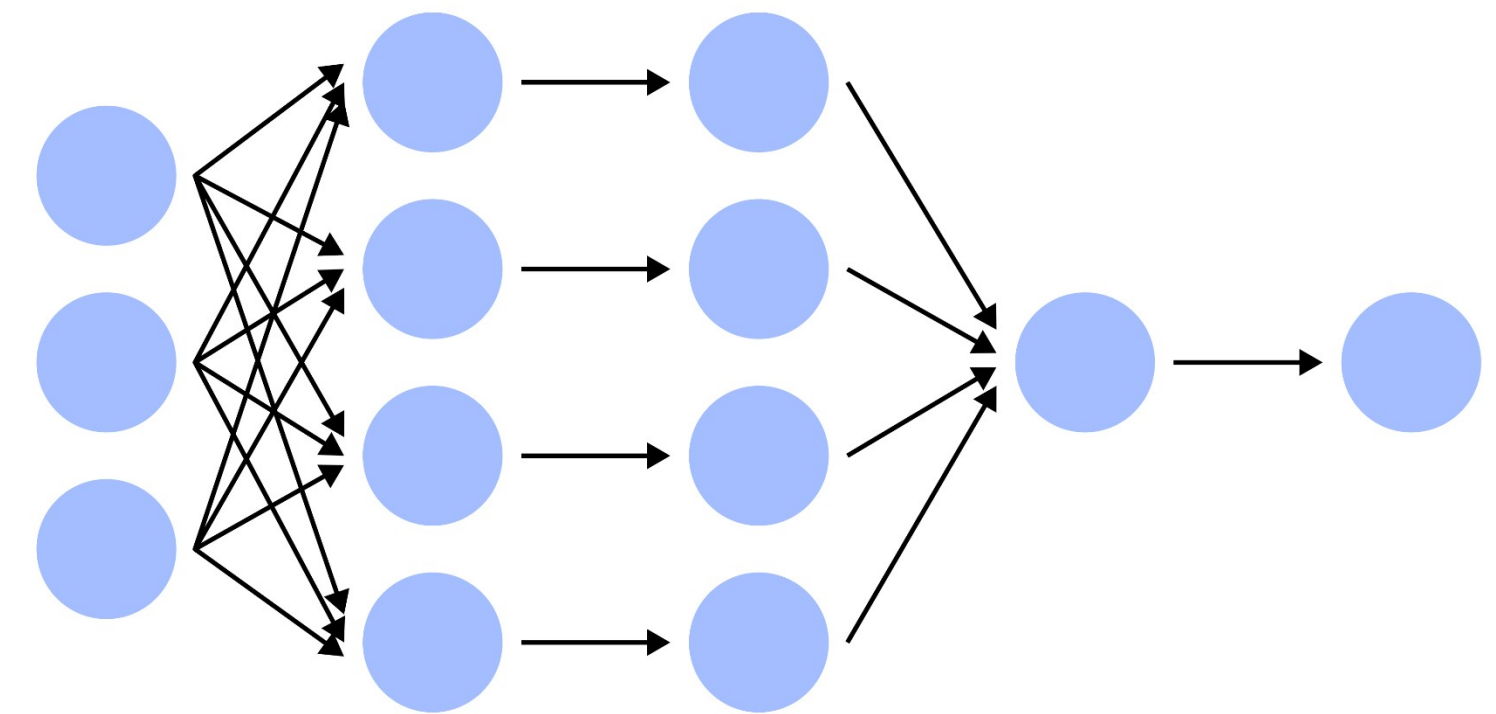
Checkpointing does not delay the data processing

Lightweight checkpointing

1/3

How it works

- Workers make up DAG, data flows from roots to leaves
- Checkpoints (barriers) are injected in root workers
- Checkpoints keep their position in an ordered data sequence
- Worker delivers a checkpoint to every outgoing edge
- Worker waits for checkpoints from all incoming edges



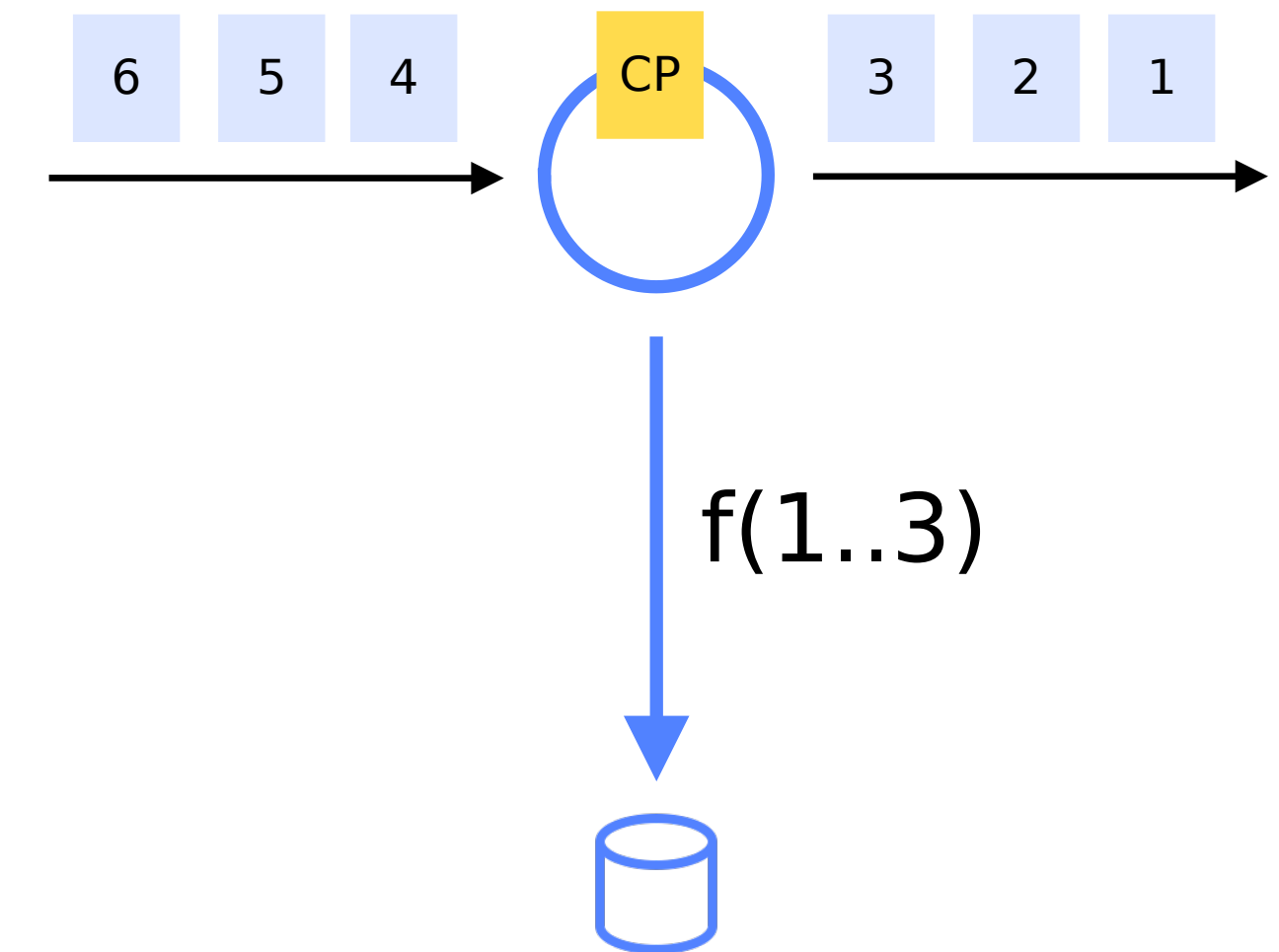
Inspired by Flink implementation (asynchronous barrier snapshotting)

Lightweight checkpointing

2/3

Saving the state

- Saved data describes the state at the time of the checkpoint
- Persisting is asynchronous (don't wait for completion)
- Root worker (source) saves an ingress stream position
- Leaf worker (sink) saves egress stream info



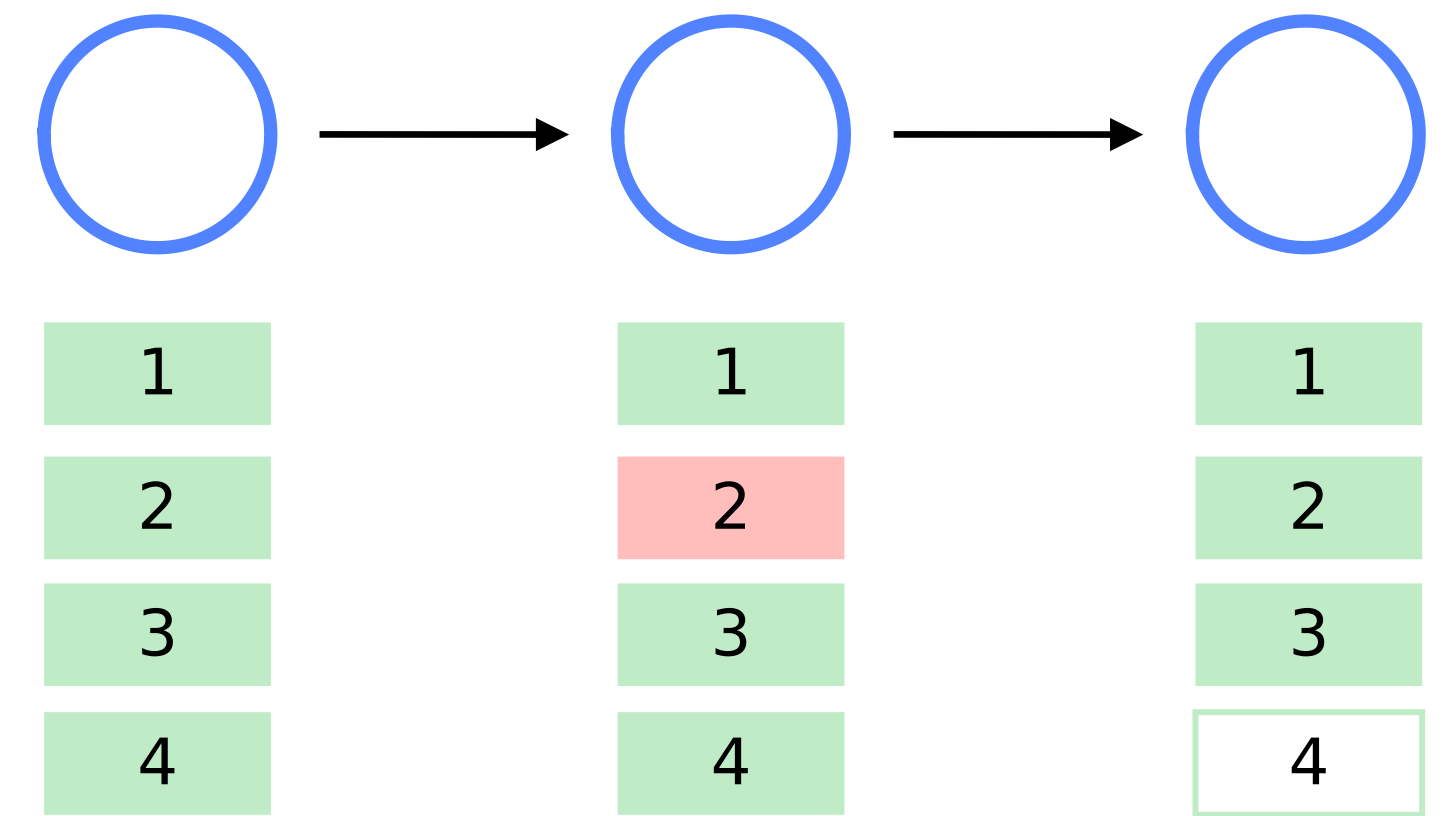
Data processing does not wait for checkpoint persistence

Lightweight checkpointing

3/3

Restart after the failure

- Worker reports to the checkpoint coordinator on successful persistence
- Coordinator marks the checkpoint as valid and completed
- Checkpointing failure does not fail request execution
- After the failure the last valid checkpoint is used



Failure of a single checkpoint doubles RPO

Security and UX

1/2

Support standard cloud practices to prevent an unauthorized access

- Single cloud-wide authentication
IAM
- Use of service accounts to access user resources
- Time-limited tokens

Compute plane is not trusted

- Sensitive data is signed in the control plane

Critical data is protected with a signature before passing to compute plane

Security and UX

2/2

Use SA

- To communicate between the system parts
- Ingress/egress data providers
- Cloud services access

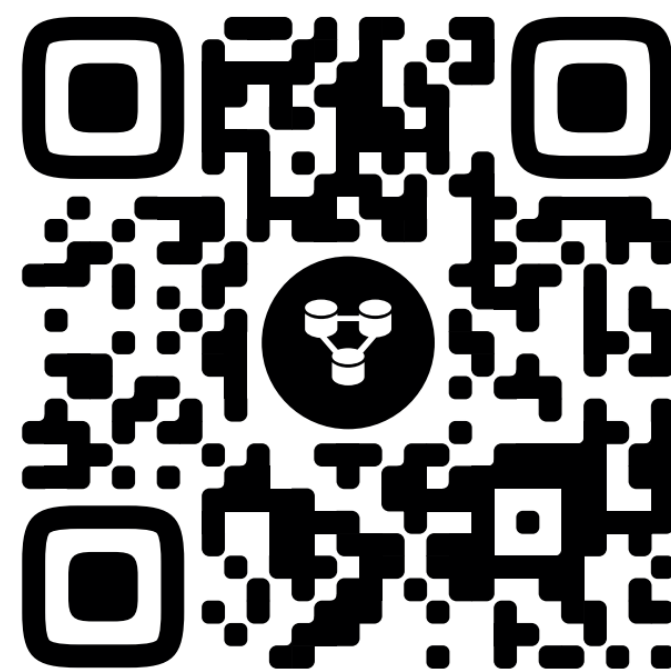
Secure presets for UX

- Connections
To hide data provider access
- Bindings (not about security, still about UX)
To hide data format

Secure and comfortable for interactive access

Leave your feedback!

You can rate the talk and give a feedback on what you've liked or what could be improved



Join our Telegram
Community Chat

QR-CODE



Co-organizer

Yandex